The Quest for Better Languages: Usage Patterns to the Rescue

Jordi Cabot

ICREA - UOC, Barcelona, Spain
jordi.cabot@icrea.cat

Abstract. Software development processes are collaborative in nature. Neglecting the key role of end-users leads to software that does not satisfy their needs. This collaboration becomes specially important when creating Domain-Specific Modeling Languages (DSMLs), which are (modeling) languages specifically designed to carry out the tasks of a particular domain. While end-users are the experts of the domain for which a DSML is developed, their contribution to the actual DSML design is, in most cases, still rather limited. This results in DSMLs that are difficult to use or that, in general, do not respond to the user needs. We argue that observing how a language is actually used in practice and deriving real usage patterns from that is the best approach to improve the language design to make it closer to what users would really like to have.

1 Introduction

Domain-Specific Modeling Languages (DSMLs) are a special kind of languages tailored to solve a particular problem in a domain. As they target a concrete domain, the development of DSMLs requires a tight collaboration between language developers and end-users, who are arguably the domain experts. While language developers provide the technical knowledge, end-users should help in setting the language concepts and shaping the notation most suitable for the domain.

Indeed, to be useful, the concepts and notation of a language should be as close as possible to the domain concepts and representation used by the end-users in their daily practice [1]. Therefore, involving end-users enriches the process and increases the chances that they will be pleased by the end result [2–4]. Unfortunately, this is an ideal scenario. In reality, it is rather common that developers work in isolation and end-users validate prototypes of the language as it evolves [5, 6].

In the last years, several approaches have proposed to make the language development process more participatory by facilitating the involvement of some end-users, either by means of example-driven approaches [7, 6] or via direct interaction/collaboration [8, 9]. This is a necessary but not sufficient condition to bridge the gap between users' needs and language experts when it comes to language design.

2 Pattern-based language definition process

Involvement of end-users early on helps to drive the design of the language but it stills has two major limitations: 1 - The set of users involved is typically small and therefore

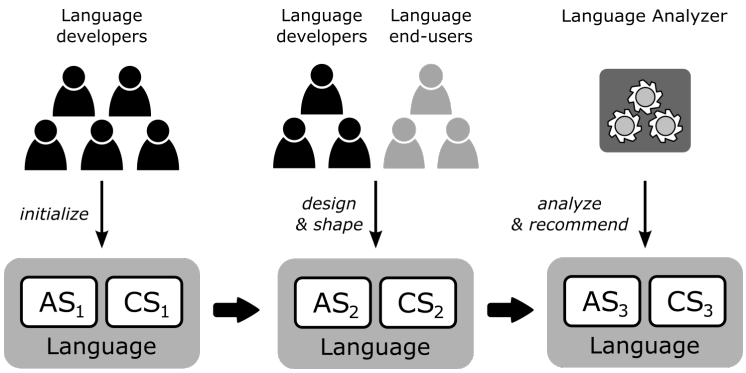

Fig. 1. Language development process. AS = Abstract Syntax. CS = Concrete Syntax.

not representative of all kinds of user profiles that will be using the language, 2 - Their feedback is based on their beliefs and opinions, which may change when they try to put the language to work later on.

Therefore, we propose to extend a language development process with a third phase, devoted to the analysis of the actual usage of the language (i.e. its *usage patterns*) once it is deployed. This process is shown in Figure 1

Phase 1 consists in developers preparing alone a preliminary version of the language. In phase 2, this version is improved thanks to the collaboration with a selected set of end-users. Once deployed, phase 3 targets the collection and analysis of language data from which a set of bottom-up patterns are inferred.

Bottom-up (or usage) patterns are generalizations of real solutions for a problem. In our context, they tell us how users adapt (or twist) the language to be able to express what they need. Therefore, by extracting and analyzing these patterns we can learn what are the real-life situations our language is not a well fit for and how we could evolve it to make sure it covers that scenario in a more natural way. This is a continuous improvement process since new problems may be uncovered at anytime (maybe even as a result of the introduction of previous solutions) or users' needs just change in the future. Figure 2 depicts this pattern-based language definition process.

The pattern identification/uncovering task can be addressed by means of corpus-based language analysis techniques [10] where repositories of language instances (i.e. repositories of models created by users employing that language) are evaluated via techniques like *instance analysis*, to determine frequency of individual elements, *relationship analysis*, to identify common co-occurrences of metamodel elements, and *clone analysis*, which seeks to identify duplicate usage of a collection of elements in the language.

This characterization of a language gives hints on possible improvements:

- Uncommon elements can be removed from the language to simplify it in a safe way
- Clusters of elements hardly ever co-occurring in the same model suggest the existence of sublanguages

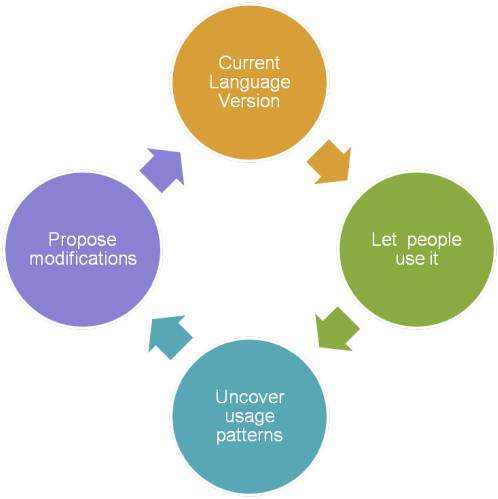

Fig. 2. Iterative language improvement.

– Complex language structures appearing often in the corpus could be replaced with new primitives expressing with a single new element the semantics of the whole structure

These modifications can then be implemented to generate a new version of the language to start the next iteration. Keeping a trace of all these language changes (and discussions since the team of language engineers may go back and forth on some of those changes before making up their mind) is important to be able to justify the language evolution at any point in the future. The Collaboro infrastructure can be used for this [8].

3 Conclusion

We have proposed a pattern-based language definition process for DSMLs that takes into account the real experiences with the language in order to improve its design. Based on the analysis of a language corpus, usage patterns can be identified and taken as the basis to suggest possible improvements.

So far, these are just hints for the language designer but it would be interesting to see how accurate they are when applied on a representative set of existing DSLs. Quite possibly, we could at least (semi)automate the process by automatically refactoring the language based on these usage patterns. This is left for further work.

Acknowledgments. Thanks to the PAME'15 Workshop oganizers (E. Syriani, R. Paige, S. Zschaler and H. Ergin) for letting me share and discuss these ideas at the workshop and to J. L. Cánovas Izquierdo and R. Tairas for all the fruitful discussions on language design over the past years.

References

1. Grundy, J.C., Hosking, J., Li, K.N., Ali, N.M., Huh, J., Li, R.L.: Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications. IEEE Trans. Softw. Eng. **39**(4) (2013) 487–515
2. Kelly, S., Pohjonen, R.: Worst practices for domain-specific modeling. IEEE Softw. **26**(4) (2009) 22 –29
3. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: Evaluating the Usability of Domain-Specific Languages. In: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments. (2012) 386–407
4. Völter, M.: MD*/DSL Best Practices (2011)
5. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-specific Languages. ACM Comput. Surv. **37**(4) (2005) 316–344
6. Cho, H., Gray, J., Syriani, E.: Creating Visual Domain-Specific Modeling Languages from End-User Demonstration. In: MiSE workshop. (2012) 29–35
7. López-Fernández, J.J., Cuadrado, J.S., Guerra, E., de Lara, J.: Example-driven meta-model development. Software and System Modeling **14**(4) (2015) 1323–1347
8. Cánovas Izquierdo, J.L., Cabot, J.: Enabling the Collaborative Definition of DSMLs. In: CAiSE conf. (2013) 272–287
9. Umuhoza, E., Brambilla, M., Ripamonti, D., Cabot, J.: An empirical study on simplification of business process modeling languages. In: SLE conf. (2015) 13–24
10. Tairas, R., Cabot, J.: Corpus-based analysis of domain-specific languages. Software and System Modeling **14**(2) (2015) 889–904